CS 543: Computer Graphics

Projection

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu

(with lots of help from Prof. Emmanuel Agu :-)

WPI
3D Viewing and View Volume

- Recall: 3D viewing set up
Projection Transformation

- View volume can have different shapes
  - Parallel, perspective, isometric

- Different types of projection
  - Parallel (orthographic), perspective, etc.

- Important to control
  - Projection type: perspective or orthographic, etc.
  - Field of view and image aspect ratio
  - Near and far clipping planes
Perspective Projection

- Similar to real world
- Characterized by *object foreshortening*
  - Objects appear larger if they are closer to camera
- Need to define
  - Center of projection (COP)
  - Projection (view) plane
- Projection
  - Connecting the object to the center of projection
Why is it Called *Projection*?

- Projectors
- Object in 3 space
- Projected image
- View plane
- COP
Orthographic (Parallel) Projection

- No foreshortening effect
  - Distance from camera does not matter
- The center of projection is at infinity
- Projection calculation
  - Just choose equal z coordinates
Field of View

- Determine how much of the world is taken into the picture
- Larger field of view = smaller object-projection size
Near and Far Clipping Planes

- Only objects between near and far planes are drawn
- Near plane + far plane + field of view = View Frustum
View Frustum

- 3D counterpart of 2D-world clip window
- Objects outside the frustum are clipped

![View Frustum Diagram]
Projection Transformation

- In OpenGL
  - Set the matrix mode to `GL_PROJECTION`
  - For perspective projection, use
    ```
    gluPerspective( fovy, aspect, near, far );
    or
    glFrustum( left, right, bottom, top, near, far );
    ```
  - For orthographic projection, use
    ```
    glOrtho( left, right, bottom, top, near, far );
    ```
gluPerspective( fovy, aspect, near, far )

Aspect ratio is used to calculate the window width

Aspect = w / h
glFrustum( left, right, bottom, top, near, far )

- Can use this function in place of gluPerspective( )
\texttt{glOrtho( left, right, bottom, top, near, far )}

- For orthographic projection
Example: Projection Transformation

```c
void display( ) {
    glClear( GL_COLOR_BUFFER_BIT );
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity( );
    gluPerspective( FovY, Aspect, Near, Far );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity( );
    gluLookAt( 0, 0, 1, 0, 0, 0, 0, 1, 0 );
    myDisplay( );    // your display routine
}
```
Projection Transformation

- Projection
  - Map the object from 3D space to 2D screen

Perspective: `gluPerspective()`

Parallel: `glOrtho()`
Parallel Projection (The Math)

- After transforming the object to eye space, parallel projection is relatively easy: we could just set all $Z$ to the same value
  - $X_p = x$
  - $Y_p = y$
  - $Z_p = -d$

- We actually want to remember $Z$ — why?
Parallel Projection

- OpenGL maps (projects) everything in the visible volume into a **canonical view volume (CVV)**

![Diagram showing the concept of canonical view volume](image)

\[ \text{glOrtho}(x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}, \text{near}, \text{far}) \]

- Projection: Need to build 4x4 matrix to do mapping from actual view volume to CVV
Parallel Projection: \texttt{glOrtho}

- Parallel projection can be broken down into two parts
  - Translation, which centers view volume at origin
  - Scaling, which reduces cuboid of arbitrary dimensions to canonical cube
    - Dimension 2, centered at origin
Parallel Projection: \texttt{glOrtho} (cont.)

- Translation sequence moves midpoint of view volume to coincide with origin
  - e.g., midpoint of $x = (x_{\text{max}} + x_{\text{min}})/2$

- Thus, translation factors are
  \[-(x_{\text{max}} + x_{\text{min}})/2, -(y_{\text{max}} + y_{\text{min}})/2, -(\text{far}+\text{near})/2\]

- So, translation matrix $M_1$:

\[
\begin{pmatrix}
1 & 0 & 0 & -(x_{\text{max}} + x_{\text{min}})/2 \\
0 & 1 & 0 & -(y_{\text{max}} + y_{\text{min}})/2 \\
0 & 0 & 1 & -(z_{\text{max}} + z_{\text{min}})/2 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
Parallel Projection: \texttt{glOrtho} (cont.)

- Scaling factor is ratio of cube dimension to Ortho view volume dimension
- Scaling factors
  \[ 2/(x_{\text{max}}-x_{\text{min}}), \ 2/(y_{\text{max}}-y_{\text{min}}), \ 2/(z_{\text{max}}-z_{\text{min}}) \]
- So, scaling matrix \( M_2 \):

\[
\begin{pmatrix}
\frac{2}{x_{\text{max}}-x_{\text{min}}} & 0 & 0 & 0 \\
0 & \frac{2}{y_{\text{max}}-y_{\text{min}}} & 0 & 0 \\
0 & 0 & \frac{2}{z_{\text{max}}-z_{\text{min}}} & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
Parallel Projection: \texttt{glOrtho}()

(\text{cont.})

- Concatenating \( M_1 \times M_2 \), we get transform matrix used by \texttt{glOrtho}

\[
\begin{pmatrix}
\frac{2}{x_{\text{max}} - x_{\text{min}}} & 0 & 0 & 0 \\
0 & \frac{2}{y_{\text{max}} - y_{\text{min}}} & 0 & 0 \\
0 & 0 & \frac{2}{z_{\text{max}} - z_{\text{min}}} & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\times
\begin{pmatrix}
1 & 0 & 0 & -(x_{\text{max}} + x_{\text{min}})/2 \\
0 & 1 & 0 & -(y_{\text{max}} + y_{\text{min}})/2 \\
0 & 0 & 1 & -(z_{\text{max}} + z_{\text{min}})/2 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

\[
M_2 \times M_1 =
\begin{pmatrix}
\frac{2}{x_{\text{max}} - x_{\text{min}}} & 0 & 0 & -(x_{\text{max}} + x_{\text{min}})/(x_{\text{max}} - x_{\text{min}}) \\
0 & \frac{2}{y_{\text{max}} - y_{\text{min}}} & 0 & -(y_{\text{max}} + y_{\text{min}})/(y_{\text{max}} - y_{\text{min}}) \\
0 & 0 & \frac{2}{z_{\text{max}} - z_{\text{min}}} & -(z_{\text{max}} + z_{\text{min}})/(z_{\text{max}} - z_{\text{min}}) \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Refer to: Hill, 7.6.2
Perspective Projection: Classical

Side view

Based on similar triangles:
\[ \frac{y}{y'} = \frac{-z}{d} \]
\[ y' = y \times \frac{d}{-z} \]
Perspective Projection: Classical (cont.)

- So \((x^*, y^*)\), the projection of point, \((x, y, z)\) onto the near plane \(N\), is given as

\[
(x^*, y^*) = \left( N \frac{P_x}{-P_z}, N \frac{P_y}{-P_z} \right)
\]

- Similar triangles

- Numerical example

Q: Where on the viewplane does \(P = (1, 0.5, -1.5)\) lie for a near plane at \(N = 1\)?

\((x^*, y^*) = (1 \times 1/1.5, 1 \times 0.5/1.5) = (0.666, 0.333)\)
Pseudo Depth Checking

- Classical perspective projection drops z coordinates
- But we **need** z to find closest object (depth testing)
- Keeping actual distance of P from eye is cumbersome and slow

\[ \text{distance} = \sqrt{(P_x^2 + P_y^2 + P_z^2)} \]

- Introduce **pseudodepth**: all we need is a measure of which objects are further if two points project to the same \((x, y)\)

\[
(x^*, y^*, z^*) = \left( N \frac{P_x}{-P_z}, N \frac{P_y}{-P_z}, aP_z + b \right)
\]

- Choose a, b so that pseudodepth varies from \(-1\) to \(1\) (canonical cube)
Pseudo Depth Checking (cont.)

- Solving:

\[ z^* = \frac{aP_z + b}{-P_z} \]

- For two conditions, \( z^* = -1 \) when \( P_z = -N \) and \( z^* = 1 \) when \( P_z = -F \), we can set up two simultaneous equations.

- Solving for \( a \) and \( b \), we get:

\[
\begin{align*}
    a &= \frac{-(F + N)}{F - N} \\
    b &= \frac{-2FN}{F - N}
\end{align*}
\]
Homogenous Coordinates

- Would like to express projection as 4x4 transform matrix
- Previously, homogeneous coordinates for the point \( P = (P_x, P_y, P_z) \) was \( (P_x, P_y, P_z, 1) \)
- Introduce arbitrary scaling factor, \( w \), so that \( P = (wP_x, wP_y, wP_z, w) \) (Note: \( w \) is non-zero)
- For example, the point \( P = (2, 4, 6) \) can be expressed as
  - \( (2, 4, 6, 1) \)
  - or \( (4, 8, 12, 2) \) where \( w=2 \)
  - or \( (6, 12, 18, 3) \) where \( w = 3 \)
- So, to convert from homogeneous back to ordinary coordinates, divide all four terms by last component and discard 4\(^{th}\) term
Perspective Projection

- Same for $x$, so we have
  
  \[
  x' = x \times d / -z \\
y' = y \times d / -z \\
z' = -d
  \]

- Put in a matrix form

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & \frac{1}{-d} & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
w
\end{pmatrix} =
\begin{pmatrix}
x' \\
y' \\
z' \\
w
\end{pmatrix} \Rightarrow
\begin{pmatrix}
-d \frac{x}{z} \\
-d \frac{y}{z} \\
-d \\
1
\end{pmatrix}
\]

OpenGL assumes $d = 1$, i.e., the image plane is at $z = -1$
Perspective Projection (cont.)

- We are not done yet!

- Need to modify the projection matrix to include $a$ and $b$

$$
\begin{vmatrix}
  x' \\
  y' \\
  z' \\
  w \\
\end{vmatrix} =
\begin{vmatrix}
  1 & 0 & 0 & 0 & x \\
  0 & 1 & 0 & 0 & y \\
  0 & 0 & a & b & z \\
  0 & 0 & (1/-d) & 0 & 1 \\
\end{vmatrix}
$$

- We have already solved $a$ and $b$
Perspective Projection (cont.)

- Not done yet! OpenGL also normalizes the x and y ranges of the view frustum to [-1, 1] (translate and scale)

- So, as in ortho, to arrive at final projection matrix
  - We translate by
    - -(xmax + xmin)/2 in x
    - -(ymax + ymin)/2 in y
  - And scale by
    - 2/(xmax – xmin) in x
    - 2/(ymax – ymin) in y
Perspective Projection (cont.)

Final projection matrix

\[
\text{glFrustum}( \text{xmin, xmax, ymin, ymax, N, F } )
\]

\[\begin{bmatrix}
\frac{2N}{\text{xmax} - \text{xmin}} & 0 & \frac{\text{xmax} + \text{xmin}}{\text{F} - \text{N}} \\
0 & \frac{2N}{\text{ymax} - \text{ymin}} & -\frac{\text{F + N}}{\text{F} - \text{N}} \\
0 & 0 & \frac{\text{F}}{\text{F} - \text{N}} \\
0 & 0 & 0
\end{bmatrix}
\]

- \( \text{N} = \) near plane, \( \text{F} = \) far plane
Perspective Projection (cont.)

- After perspective projection, viewing frustum is also projected into a canonical view volume (like in parallel projection)